Esercitazione Processi

Laboratorio Sistemi Operativi

Aniello Castiglione
Email: aniello.castiglione@uniparthenope.it

Text editor vi

Text editor *vi*

- Per scrivere un file in Unix potete usare uno degli editor disponibili: "pico", "gvi", "vi“
- pico e gvi sono editor *moderni,* in cui potete manipolare il testo usando mouse e frecce. gvi accetta anche tutti i comandi di vi
- vi é l'editor di default in Unix. E' difficile da usare, perché è stato sviluppato quando non esistevano le moderne interfacce grafiche, e il mouse non era ancora stato inventato
- vi é molto potente, e i suoi comandi possono persino essere messi in un file ed eseguiti separatamente
- VIM sta per VIsual editor iMproved, cioe' editor visuale migliorato. vim permette tutte le funzionalità di VI ed in più contiene numerose migliorie

Text editor *vi*

- In vi esistono due modalità:
  - COMANDO, usata per spostarsi nel testo, cancellare, duplicare, sostituire, cercare, etc.
  - INSERZIONE, per inserire testo
  - Si passa da una modalità all'altra con opportuni comandi

```
$ vi prova.c  /* crea o apre prova.c */
```

- quando editate un file, inizialmente siete in modalità COMANDO

Text editor *vi*

- MODO "COMANDO": come uscire dall'editor
  - **:wq** chiude l'editor salvando le modifiche
  - **:w** salva le modifiche senza chiudere
  - **:q** chiude l'editor senza salvare le modifiche
  - **:q!** chiude l'editor senza salvare le modifiche

Text editor *vi*

- MODO "COMANDO": come spostarsi nel file
  - **h** si sposta alla sinistra di un carattere
  - **l** si sposta alla destra di un carattere
  - **j** si sposta sotto di una linea
  - **k** si sposta sopra di una linea
  - **ctrl-u** si sposta sopra di alcune linee
  - **ctrl-d** si sposta sotto di alcune linee
  - **G** si sposta alla fine del file
  - ***n*G** si sposta alla linea numero *n* del file
  - **$** si sposta alla fine della riga
  - **^** si sposta all'inizio della riga
  - ***:1*** si sposta all'inizio del file
  - ***:20*** si sposta alla riga 20

Text editor "vi"

- MODO "COMANDO": come cancellare
  - **x** cancella il carattere attuale
  - **dd** cancella la linea attuale
  - **dw** cancella la parola attuale
  - ***n*x** cancella *n* caratteri (verso destra) a partire dal cursore
  - ***n*dd** cancella *n* linee in avanti
  - ***n*dw** cancella *n* parole a partire dalla destra del cursore

Text editor "vi"

- MODO "COMANDO": altri comandi utili
  - ***r*x** rimpiazza l'attuale carattere con il carattere *x*
  - **.** ripete l'ultimo comando
  - **u** undo l'ultimo comando
  - ***/string*** ricerca la prima occorrenza della stringa *string* nel file in avanti
  - ***?string*** ricerca la prima occorrenza della stringa *string* nel file all'indietro
  - **n** dopo un comando di ricerca, ripete la ricerca

Text editor *vi*

- MODO "COMANDO": altri comandi utili
  - ***n*yy** copia *n* linee in un buffer locale. Utile per copiare linee in parti differenti del testo
  - **p** pone il contenuto del buffer a partire dalla posizione del cursore
  - il comando **p** può essere usato per inserire il contenuto del buffer locale del vi. Questo buffer viene riempito non solo dal comando **yy**, ma anche, ogniqualvolta si usa un comando di cancellazione, con la parte di testo cancellata. Così è possibile spostare porzioni di testo da una parte all'altra del file

Text editor *vi*

- MODO "COMANDO": altri comandi utili
  - **%** se il cursore si trova su una parentesi, posiziona il cursore alla corrispondente parentesi di bilanciamento.
    - E' utile quando si scrivono programmi in C per controllare il bilanciamento delle parentesi dei costrutti
  - **:N,Ms/*OLD*/*NEW*/g** sostituisce ogni occorrenza della stringa *OLD* con la nuova stringa *NEW* tra le righe N e M

Text editor *vi*

- MODO "COMANDO": altri comandi utili
  - **:r FILENAME**
    - inserisce il contenuto di FILENAME nel file attuale a partire dalla posizione del cursore
  - **:set number**
    - abilita la numerazione delle linee del file
  - **:set nonumber**
    - rimuove la numerazione delle linee del file

Text editor *vi*

- Passaggio dal modo "COMANDO“ al modo "INSERZIONE“
  - **i** inserisce il testo a sinistra della posizione del cursore
  - **a** appende il testo a destra della posizione del cursore
  - **o** crea una nuova linea al di sotto della posizione del cursore
  - **O** crea una nuova linea al di sopra della posizione del cursore
  - **S** sostituisce un carattere dalla posizione del cursore con un testo nuovo
  - **ns** sostituisce **n** caratteri dalla posizione del cursore con testo nuovo
  - **cw** cambia la parola

Text editor *vi*

- Passaggio dal modo "INSERZIONE“ al modo "COMANDO"
  - **<esc>**
- In ogni momento, per passare dalla modalità INSERZIONE a quella di comando, basta digitare il tasto ESCAPE

Processi

ESERCIZI

Esempio

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (int argc, char *argv[]) {
pid_t childpid = 0;
int i, n;
if (argc != 2){ /* Controllo argomenti*/
fprintf(stderr, "Uso: %s processi\n", argv[0]);
return 1;
}
n = atoi(argv[1]);
for (i = 1; i < n; i++)
if ((childpid = fork()) <= 0)
break;
fprintf(stderr, "i:%d processo ID:%d padre ID:%d figlio
ID:%d\n",
i, getpid(), getppid(), childpid);
return 0;
}
```

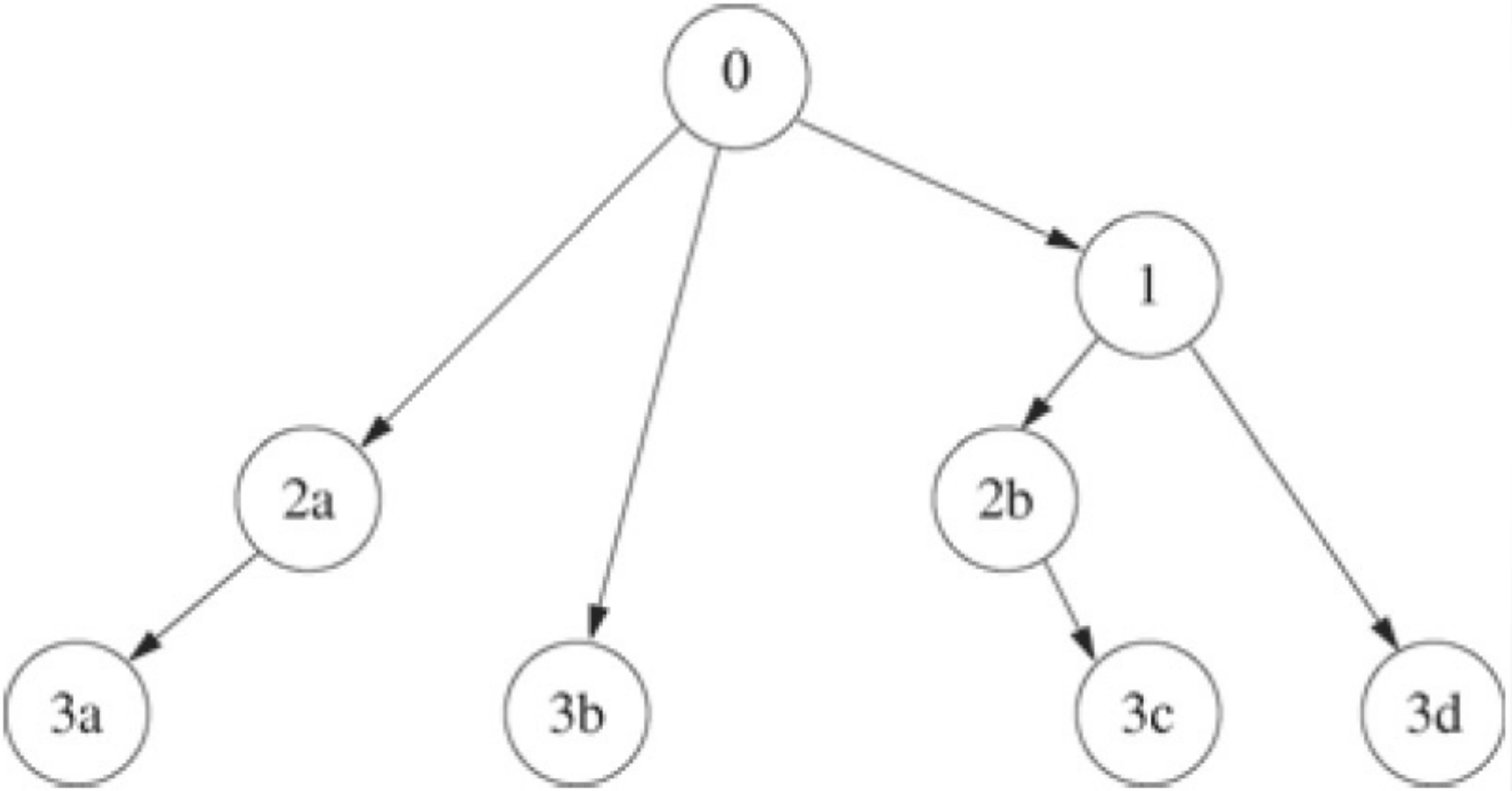
Osservazioni

- Cosa succede se sostituiamo il test

  (childpid = fork()) <= 0 con (childpid = fork()) == -1

- **Risposta:**
- Tutti I processi restano nel ciclo a meno che la fork fallisca. Ogni iterazione del ciclo raddoppia il numero di processi, formando un albero del tipo riportato in figura per n=4
  - In figura, ogni processo è rappresentato con un cerchio la cui etichetta è il valore di i al momento della sua creazione
  - Il processo originario ha etichetta 0
  - Le lettere distinguono processi creati con lo stesso valore di i
- In questo programma non si distingue tra padre e figlio dopo la fork
  - Entrambi procedono a creare figli alla successiva iterazione del ciclo

Esempio: albero dei processi creati (n=4)



Esercizio 1

```
int glob=5;
int pid=0;
pid=fork();
glob--;
pid=fork();
glob--;
if (pid!=0) {
     pid=fork();
     glob--;
}
printf("Valore di glob=%d\n",glob);
```

Esercizio 2

```
int glob=5;
int pid=0;
int main() {
      int i=0;
      for (i=1;i<3;i++) {
            pid=fork();
                  if (pid==0) {
                        glob=glob*2;
                        sleep(i+1);
                  }
            glob=glob+1;
            printf("Valore di glob=%d\n",glob);
}
}
```

Esercizio 3

- Scrivere un programma C che:
  - Crea un processo figlio, stampa il messaggio “In attesa” ed attende la terminazione del figlio.
  - Il figlio esegue il comando “ls -l”
  - Quando il figlio termina, il padre visualizza il messaggio “nuovo figlio” e crea un secondo processo figlio.
  - Il secondo figlio aspetta per 5 secondi, stampa a video un messaggio e termina.
  - Quando il processo figlio termina, il padre stampa a video il pid del processo terminato.

Esercizio 4

- Scrivere un programma C in cui un processo crea un processo figlio
  - Il processo figlio calcola la sequenza di Fibonacci di ordine n (n<=12). Quando termina restituisce il valore calcolato come codice di terminazione
  - Il padre attende la terminazione del figlio ed esamina lo stato di terminazione
    - Se lo stato di terminazione è relativo ad una terminazione con successo e il codice di terminazione è un valore minore di 50
      - Crea un secondo figlio che esegue il comando ls –al a.out
      - Attende il secondo figlio, stampa un messaggio e termina
    - Altrimenti, stampa un messaggio e termina

Esercizio 5

- Realizzare un programma in C e Posix sotto Linux che realizzi una struttura di processi ad albero ternario, tale che ogni processo si metta in attesa che i suoi figli terminino. Ogni figlio termina dopo aver atteso per un numero di secondi pari al livello dell'albero al quale si trova, allo scadere del quale stampa a schermo la stringa "Concluso! e comunica la genitore la sua terminazione.